APPENDIX K: ADVANCED PUBLISHER .MSI PACKAGING

30 Jun 2004

NMCI Packaging Standards

Best Practices for MSI

DRAFT Version 1.0
15 January 2004

Packaging Standards / Best Practices for MSI

Final

Version 1.0

May 27, 2004

NMCI.90149.01.F+0.E

Revision History

The Revision History table below lists in chronological order each minor revision of this document. A minor revision is defined as a set of changes affecting fewer than 30 percent of the pages in the document.

Date	Author	Revision Number	Change(s) Made	Affected Page(s)

Entries in the Revision History table are deleted when a document undergoes a major revision, called a document update. A document update is defined as a set of changes affecting more than 30 percent of the pages in the document. Document updates do not need to be listed in the Revision History table.
For more information about Navy Marine Corps Intranet (NMCI) documentation, contact the manager of the Document Management Center (DMC) at ISFDOCSMailbox@eds.com.

Document Storage

The EDS NMCI Operations Library assigns identification (ID) numbers for NMCI documents and stores the master Microsoft Word editions as well as the PDF versions. The librarian assigns a document ID number and maintains an audit trail of all document versions. To contact the library, telephone or e-mail the EDS NMCI Operations Librarian (James R. Taylor, 703-742-1940, ISFOPSLibrary@eds.com).

Table of Contents

2 GENERAL DEVELOPER AND PACKAGER GUIDELINES

The **NMCI Release Development and Deployment Guide** (NRDDG) provides general guidelines for application development. These standards already apply to CDAs; however, to clarify MSI Packaging Standards, the following pertains:

- These general guidelines for application development are provided for reference. Referenced sources for this document are from Microsoft, Novadigm, and Chapter 4 of the NRDDG.
- Successful application deployment depends not only on how applications are packaged and installed, but also on how applications are architected. The general recommendations in this document affect how applications operate when installed, and allow for applications to be successfully deployed in Enterprise environments.

2.1 APPLICATION ARCHITECTURE

Applications should be designed to use machine components, such as file and Registry entries, that are identical for all users. User preferences, such as icons, settings, and Registry values, should be stored such that one user does not and cannot affect another user's settings.
Machine components are global settings for the machine shared by all users of the application.

- Machine components affect all users.
- Applications should not require users to have write permission to machine components.
- Machine components should be able to be deployed when no user is logged onto a workstation.
- Machine components include HKEY_LOCAL_MACHINE, shortcuts and settings stored in the All Users profile.
- Application files generally are to be found under the "Program Files" directory.
- Applications should avoid using HKEY_USERS\.DEFAULT as a repository for configuration information. Applications running as system services should utilize HKEY_LOCAL_MACHINE for settings.

User components are specific to each user of the application. The application should be able to initialize user settings upon first launch either internally to the application or via advertised shortcuts.

- User settings should be stored in the HKEY_CURRENT_USER or within the user's profile.
- User settings should support Roaming Profiles.
- A user's preferences should not affect another user's preferences.

2.1.1 Machine and User Components

- The MSI package should support the ability to fully deploy necessary components (machine components) during an unattended installation, even when no user is logged onto the workstation.
- Windows Installer should install user components at the time of application installation if the user is logged onto the workstation or upon first application launch via advertised shortcuts,

which will validate key files and Registry entries to ensure that the user components are initialized before the application is executed. If the user settings do not need to be initialized to a particular customized value, user components could be created by the application when launched.

- The ALLUSERS=1 property should be used to assure proper deployment of advertised shortcuts to the “All Users” profile when applications are to be deployed during an unattended nightly process but available to all users once they logon.
- If an application is expected to be available to only one user, then ALLUSERS=”” will install the application only for the user. Windows Installer may require the user have elevated privileges in order to install the application.

2.1.2 Package Composition

- Use logical directories and root points within the MSI. This allows for the MSI to be portable across configurations.
  - Example: [ProgramFilesFolder] vs. “C:\Program Files” will support users who utilize a second drive partition with the path “D:\Program Files”. For example, applications that need to run on terminal servers may need to support an alternate system drive other than “C:” and should support automatic use of the system drive or logical directory.
- Advertise Registry keys and shortcuts to support self-healing.
- When significant differences exist between a platform or operating system build (image) versions of an application, it is recommended that separate packages be created for deployment to each platform or build. This allows for an automation system to control and limit, on a platform basis, which MSI package is to be deployed.
- Where platform, OS, or image-specific versions of an application only differ by a few components or settings, a single MSI can be created to support the multiple platforms, provided those changes are relatively small in size by comparison.

2.1.3 Package Materials and Administrative Installation Points

In order to efficiently distribute application components across a network, all MSI packages should utilize an Administrative Installation Point (AIP). An AIP allows the application source to be located on a network share or behind a web server. Windows Installer will download only the necessary components based upon the MSI package from the AIP.

- Avoid using multiple MSIs to do what a single MSI could accomplish. Use a single MSI for installing the entire application—however, each MSI should only contain a single “product or application.”
- The MSI should support creation of an Administrative Installation Point (AIP) in order to allow Windows Installer to transfer only necessary data for the installation.
- Packages should be extracted to AIPs in order to minimize network bandwidth utilization during application component downloads for installation and repair. Embedding large CAB files within the MSI or even deploying large CAB files that are external to the MSI is less efficient because unused components will be transferred across the network.

- Limit the size of a single Cabinet file. There is a general CAB file limitation of 65,536 contiguous file resources contained within a single CAB file. If an application requires a larger number of files, multiple CAB files will be required.
- Some packages may need to reference additional files for custom actions or scripts. These additional components should be included within the MSI so they can be managed by the MSI directly on the target system. They should not be required to be standalone components that were not deployed as part of the MSI package. (Do not allow custom scripts or actions to launch in the current user context. They need to stay with the installation user so they have the permissions of the System account).

  NOTE: It may acceptable for a package to reference an external DLL or file that is not delivered as part of the package but is expected to already be present on the target system. Standards should be developed regarding external files or components and their locations. If the files are not included within the MSI as a managed component, they should not be located within the MSI directory or sub-directory.
- The MSI should support valid short-file names, which can optimize transfers in Enterprise environments. This does not mean that the entire program itself should use short file names. Rather, the MSI should include short-filename mappings where required to support AIP creation. The MSI should be able to create the AIP with valid short file names. This is done by populating the File, Shortcut, and Directory tables with both short and long file names in the format "short|long file name."

3 MSIEXEC COMMAND AND APPROPRIATE SWITCHES

The following table depicts the "msiexec" command and appropriate switches.
The executable program that interprets packages and installs products is Msiexec.exe. Note that Msiexec also sets an error level on return that corresponds to system error codes. The following table describes the command-line options for this program:

Option	Parameters	Meaning
/I	*Package*|*ProductCode*	Installs or configures a product.
/f	[p|o|e|d|c|a|u|m|s|v] *Package*|*ProductCode*	Repairs a product. This option ignores any property values entered on the command line. The default argument list for this option is 'pecms'. This option shares the same argument list as the REINSTALLMODE property. p - Reinstalls only if file is missing. o - Reinstalls if file is missing or an older version is installed. e - Reinstalls if file is missing or an equal or older version is installed. d - Reinstalls if file is missing or a different version is installed. c - Reinstalls if file is missing or the stored checksum does not match the calculated value. Only repairs files that have msidbFileAttributesChecksum in the Attributes column of the File table. a - Forces all files to be reinstalled. u - Rewrites all required user-specific Registry entries. m - Rewrites all required computer-specific Registry entries. s - Overwrites all existing shortcuts. v - Runs from source and recaches the local package. Do not use the v reinstall option for the first installation of an application or feature.
/a	*Package*	Administrative installation option. Installs a product on the network.
/x	*Package*|*ProductCode*	Uninstalls a product.
/j	[u|m]*Package* *or* [u|m]*Package* /t *Transform List* *or* [u|m]*Package* /g *LanguageID*	Advertises a product. This option ignores any property values entered on the command line. u - Advertises to the current user. m - Advertises to all users of machine. g - Language identifier. t - Applies transform to advertised package.

Option	Parameters	Meaning
/L	[i|w|e|a|r|u|c|m|o|p|v|x|+|!]*Logfile*	Specifies path to log file. Flags indicate which information to log. i - Status messages. w - Nonfatal warnings. e - All error messages. a - Start up of actions. r - Action-specific records. u - User requests. c - Initial UI parameters. m - Out-of-memory or fatal exit information. o - Out-of-disk-space messages. p - Terminal properties. v - Verbose output. x - Extra debugging information. Only available on Windows Server 2003. + - Append to existing file. ! - Flush each line to the log. "*" - Wildcard, log all information except for the v and x options. To include the v and x options, specify "/l*vx".
/m	*filename*	Generates an SMS status .mif file. Must be used with either the install (-i), remove (-x), administrative installation (-a), or reinstall (-f) options. The ISMIF32.DLL is installed as part of SMS and must be on the path. The fields of the status mif file are filled with the following information: Manufacturer - Author Product - Revision Number Version - Subject Locale - Template Serial Number - not set Installation - set by ISMIF32.DLL to "DateTime" InstallStatus - "Success" or "Failed" Description - Error messages in the following order: 1) Error messages generated by installer. 2) Resource from Msi.dll if installation could not commence or user exit. 3) System error message file. 4) Formatted message: "Installer error %i", where %i is error returned from Msi.dll.
/p	*PatchPackage[;patchPackage2…]*	Applies a patch. To apply a patch to an installed administrative image you must combine options as follows:

Option	Parameters	Meaning
		/p *<PatchPackage>[;patchPackage2…]* /a *<Package>*
/q	n|b|r|f	Sets user interface level. q , qn - No UI qb - *Basic UI* . Use qb! to hide the Cancel button. qr - *Reduced UI* with no modal dialog box displayed at the end of the installation. qf - *Full UI* and any authored FatalError, UserExit, or Exit modal dialog boxes at the end. qn+ - No UI except for a modal dialog box displayed at the end. qb+ - Basic UI with a modal dialog box displayed at the end. The modal box is not displayed if the user cancels the installation. Use qb+! or qb!+ to hide the Cancel button. qb- - Basic UI with no modal dialog boxes. Please note that /qb+- is not a supported UI level. Use qb-! or qb!- to hide the Cancel button. Note that the ! option is available with Windows Installer version 2.0 and works only with basic UI. It is not valid with full UI.
/? or /h		Displays copyright information for Windows Installer.
/y	*module*	Calls the system API DllRegisterServer to self-register modules passed in on the command line. For example, msiexec /y MY_FILE.DLL. This option is only used for Registry information that cannot be added using the Registry tables of the .msi file.
/z	*module*	Calls the system API DllUnRegisterServer to unregister modules passed in on the command line. For example, msiexec /z MY_FILE.DLL. This option is only used for Registry information that cannot be removed using the Registry tables of the .msi file.
/c		Advertises a new instance of the product. Must be used in conjunction with /t. Available starting with the Windows Installer version shipped with the Windows Server 2003 family and Windows XP SP1.
/n	*ProductCode*	Specifies a particular instance of the product. Used to identify an instance installed using the multiple instance support through a product code changing transforms. Available starting with the Windows Installer version shipped

Option	Parameters	Meaning
		with the Windows Server 2003 family and Windows XP SP1.

The options /i, /x, /f[p|o|e|d|c|a|u|m|s|v], /j[u|m], /a, /p, /y and /z should not be used together. The one exception to this rule is that patching an administrative installation requires using both /p and /a. The options /t, /c and /g should only be used with /j. The options /l and /q can be used with /i, /x, /f[p|o|e|d|c|a|u|m|s|v], /j[u|m], /a, and /p. The option /n can be used with /i, /f, /x and /p.
To install a product from A:\Example.msi, install the product as follows:

```
msiexec /i A:\Example.msi
```

Only public properties can be modified using the command line. All property names on the command line are interpreted as uppercase but the value retains case sensitivity. If you enter **MyProperty** at a command line, the installer overrides the value of MYPROPERTY and not the value of **MyProperty** in the Property table. For more information, see About Properties.
Steps that NMCI Certification Engineers use to process the Site/Application Developer- Provided MSI:

1. Adhere to the Novadigm Checklist for Publishing Windows Installer Applications via Radia (http://techsupport.novadigm.com/kb/kb01033.asp).
2. Full MSI validation using Microsoft's Orca utility (there should be no Errors or Warnings).
3. Create the ACP/AIP (normally placed in the C:\AIP\ApplicationName or C:\AIP\InstanceName subdirectory).

   ```
   msiexec /a application.msi SHORTFILENAMES=true /l*v c:
   ```
4. Install the application from the AIP

   ```
   msiexec /i application.msi /qb ALLUSERS=1 /l*v C:\ACP\ApplicationName\AppInstallMSI.log
   msiexec /i application.msi /qr ALLUSERS=1 /l*v C:\ACP\ApplicationName\AppInstallMSI.log
   msiexec /i application.msi /qf ALLUSERS=1 /l*v C:\ACP\ApplicationName\AppInstallMSI.log
   msiexec /i application.msi /qn ALLUSERS=1 /l*v C:\ACP\ApplicationName\AppInstallMSI.log
   ```
   NOTE: The /l*v C:\ACP\ApplicationName\AppInstallMSI.log enables logging (l) verbose(v) location and filename of the log file.
5. Remove the application using the application.msi from the AIP created in Step 2

   ```
   msiexec /x application.msi /qb
   ```

To install a product with PROPERTY set to VALUE use the following syntax on the command line. You can put the property anywhere except between an option and its argument.

Correct syntax:

```
msiexec /i A:\Example.msi PROPERTY=VALUE
```

Incorrect syntax:

```
msiexec /i PROPERTY=VALUE A:\Example.msi
```

Property values that are literal strings must be enclosed in quotation marks. Include any white spaces in the string between these marks.

```
msiexec /i A:\Example.msi PROPERTY="Embedded White Space"
```

To clear a public property using the command line, set its value to an empty string.

```
msiexec /i A:\Example.msi PROPERTY=""
```

For sections of text set apart by literal quotation marks, enclose the section with a second pair of quotation marks.

```
msiexec /i A:\Example.msi PROPERTY="Embedded ""Quotes"" White Space"
```

The following is an example of a complicated command line.

```
msiexec /i testdb.msi INSTALLLEVEL=3 /l* msi.log COMPANYNAME="Acme ""Widgets"" and
""Gizmos."""
```

The following example illustrates advertisement options. Note that switches are not case sensitive.

```
msiexec /JM msisample.msi /T transform.mst /LIME logfile.txt
```

This example shows how to install a new instance of a product to be advertised. This product has been authored to support multiple instance transforms.

```
msiexec /JM msisample.msi /T :instance1.mst;customization.mst /c /LIME logfile.txt
```

This example shows how to patch an instance of a product that has been installed using multiple instance transforms.

```
msiexec /p msipatch.msp;msipatch2.msp /n {00000001-0002-0000-0000-624474736554} /qb
```

When applying patches to a particular product, the /i and /p options cannot be specified in a command line together. In this case, you can apply patches to a product as follows.

```
msiexec /i A:Example.msi PATCH=msipatch.msp;msipatch2.msp /qb
```

NOTE: The PATCH property cannot be set in command line, when /p option is used. If the PATCH property is set when the /p option is used, the value of PATCH property is ignored and overwritten.

4 ENTERPRISE PACKAGING STANDARDS AND TEMPLATES

- To assure package consistency and productivity, standard packaging processes and templates should be used.
- Templates should incorporate the desired default settings for all packages created within the packaging area - such as permissions, state file filters, library filters, as well as displaying a standard set of bitmaps or banners associated with the packaging area. This will ensure the MSIs created are consistent and visually recognizable as products of a particular packaging area.
- Enterprises should use templates to automate settings for package creation in order to ensure consistency and help eliminate error and omissions in the package creation process.
- Templates should be developed that are consistent with Enterprise standards for directory structure, application locations, security and permissions, etc. that are compatible with the target build environments.

NOTE: Standards will vary from Enterprise to Enterprise and may also vary within a single Enterprise depending on the target environments being managed. For example, file locations on Windows 2000 versus Windows XP or file locations on desktops versus laptops. Templates and automation should be used to eliminate variance from the standards.

5 SUPPORTING MSI UPGRADES

Proper MSI coding techniques dictate that the MSI should support upgrading. This is the ability to uninstall, reinstall or install over or along side of different versions of the same application. When building an MSI package, an author should be familiar with how the MSI upgrading process works to ensure that their packages are compliant with the best practices for application upgrading. In other words, a package upgrade strategy should be incorporated into the first build of an application package. Windows Installer has package upgrade logic built into the MSI File and Properties tables. Several items must be taken into account to ensure proper upgrading of an application. The following MSI properties help control the upgrade logic:

- **PackageCode**: The PackageCode is actually contained in the MSI file's Summary information, rather than the Property table. It is designed to give each package a unique code. The PackageCode is used like an MD5 hash value to uniquely identify an MSI package.
- **ProductCode**: The ProductCode is a required row of the MSI Property table that uniquely identifies the Product (application) being installed.
- **ProductVersion**: The ProductVersion is also a required row of the MSI Property table. It uniquely identifies the version of the Product being installed in the form Major.Minor.Build.Release. The Windows Installer Engine only uses the first three of these numbers, effectively ignoring the Release number.
- **ProductLanguage**: Another required row of the Property table within the MSI that defines the language for the package using LANGID values (i.e. 1033 for US English).
- **UpgradeCode**: UpgradeCode is an optional row of the Property table that identifies packages that belong in the same Product upgrade group. Though it is an optional row, an UpgradeCode should be entered in preparation for future upgrades.

There are three types of upgrades within Windows Installer: Small, Minor, and Major:

- **Small Upgrade**: The ProductCode and ProductVersion are the same in the older and newer packages. Only the PackageCodes are different. Because the ProductVersion is not changed, there is no way to detect which package contains the newer version of the Product and this type of upgrade is seldom used.
- **Minor Upgrade**: The ProductCode is the same, but the ProductVersion is higher in the newer version. Minor Upgrades will remove existing installations of the Product with a lower version number.
- **Major Upgrade**: With Major Upgrades, the ProductCodes are different and the ProductVersion is higher in the upgrade package, but the UpgradeCodes are the same.

Another concern when upgrading an application is the preservation of customizations made by the user. User customizations can appear within a file or in the Registry.

By default, Windows Installer performs file versioning to determine when a file should be updated with the file in the package. Non-versioned files are considered "User Data"—meaning Windows Installer will compare the Create and Modified dates on the file, and if the Modified date is later than the Create date, the existing file is not replaced, assuming it would remove user customizations in the process.

For customizations maintained in the Registry or INI files, the entry should be set to not replace if it exists. This allows default values to be set if they do not previously exist while preserving user changes to the setting if they do exist.

- Adhere to Microsoft standards for patching and upgrading.
  http://msdn.microsoft.com/library/en-us/msi/setup/patching_and_upgrades.asp
- The reference to the Microsoft article on upgrading to a new version of an application can be found in the "Microsoft Platform SDK February 2003 – "Preparing an Application for Future Major Upgrades"
  http://msdn.microsoft.com/library/en-us/msi/setup/preparing_an_application_for_future_major_upgrades.asp

6 UNATTENDED ENTERPRISE DEPLOYMENT GUIDELINES

In order to support Enterprise deployment of applications, an automated system will require that MSI packages provide parameters and options to allow administrators to control the manner in which the packages are installed, updated, and removed from systems.

6.1 RUN INSTALLATION UNATTENDED / ATTENDED

Enterprise software management generally requires the ability to perform unattended installation and removal of applications during off-peak hours. Each MSI package should have the ability to run the installation / un-installation un-attended. Unattended installations require the system to be able to provide any configuration information as part of the package installation. If such information is not provided or is user/seat specific, such as the information provided via the user interface during a manual installation, the MSI package should prompt the user during the first launch of the application.
Two areas from the Microsoft Platform SDK are of value:

- "**Command Line Options**" - MSIs should have the ability to run the installation / un-installation quietly. Best Practices would be to use the Microsoft standard MSI command line switches.

  http://msdn.microsoft.com/library/en-us/msi/setup/command_line_options.asp
- Administrators can set the UI level using the command line option "/q". A complete list of these command line options can be referenced under the "Using the User Interface" section.

  http://msdn.microsoft.com/library/en-us/msi/setup/using_the_user_interface.asp

6.2 SUPPRESSION OF REBOOT AND THE FORCE REBOOT PROPERTY

The MSI package should support the standard command line options for suppression of the reboots during an installation process. Deferred reboots are generally required to allow an automated distribution system to manage when and how the target nodes are restarted. This may also be required in order for administrators to arrange the sequence an MSI package deployment with other package deployments to the target system.
The MSI package should allow the automation system to manage reboots of the machine when they are required, as other processing may need to occur before or after the MSI package installation.
The package should utilize the MSI REBOOT Property (Force, Suppress, and ReallySuppress), which handles the rebooting of the MSI. If a reboot is required, Windows installer can notify the automation system via return codes 1641 and 3010. The Microsoft Reference for this property is:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msi/setup/rebootprompt_property.asp

NOTE: This is a standard Windows Installer command that can be run with any MSI, so re-packagers do not have an option about utilizing it.

Application packages should not initiate reboots via a custom action. Custom Action reboots can interrupt both the Windows Installer processing as well as an Enterprise automation system.

6.3 PROPERTY OVERRIDES

Per the Microsoft Platform SDK, Windows Installer uses three categories of global variables during an installation:

- **Private properties:** The installer uses private properties internally and their values must be authored into the installation database or set to values determined by the operating environment.
- **Public properties:** Public properties can be authored into the installation database in the same way as private properties. In addition, a user or system administrator can change the values of public properties by setting the property on the command line, by applying a transform, or by interacting with an authored user interface. Public property names cannot contain lowercase letters.
- **Restricted public properties:** For security purposes, the author of an installation package can restrict the public properties a user can change.

Not all properties need to be defined in every package. There is a small set of required properties that must be defined in every package. The installer sets the values of properties in a particular order of precedence.

For Enterprise management of any MSI, the settings required by the application should be configured using properties, and any values required during installation should be exposed as public properties or prompted for upon the first launch of the application to facilitate scripted or unattended installation. Examples of properties include application paths, license strings, application host and port settings, etc.

7 GUIDELINES FOR NON-MSI PACKAGE RE-AUTHORING

For Enterprise re-packaging of non-MSI application installations into MSI format, several recommendations should be followed in addition to those provided for general application development and packaging guidelines:

- The packaging environment should ensure that packages are built on identical equipment and operating system builds (images) as those in use in production.
- The packaging environment should be configured such that it will be refreshed to a baseline image state prior to each new packaging operation. Ideally, the packaging tool will allow for saving the baseline state as well as the image in order to speed up the scanning process.
- The packaging tool should provide features to automate and adhere to the standards described above. In addition, it should support automation and the use of standardized templates for the enforcement of Enterprise packaging standards for non-MSI package re-authoring as MSIs.
- The packaging tool should provide a wizard-driven process to facilitate a simple and consistent package creation process.
- The packaging tool should automatically populate public and private properties from a variety of different sources such as string substitution, transforms, Registry keys and values, .INI files, environment variables, conditional expressions, and user interface prompts.
- The packaging tool should provide the ability to create a package in the following methods:
  - Component selection mode where the GUI provides an easy way to select just the Registry and file data the administrator wishes to become part of a package.
  - Repackage from an Installation CD where the administrator performs a before and after capture of what the installation deploys to the desktop.
  - Active monitoring of application execution to determine what it does to the desktop as it executes and records the components that are referenced.
- The packaging tool should provide for extensive component filtering to in order to exclude undesired components from the packages. Ideally these filters will be configurable as template settings that can be used across the Enterprise.
- The packaging tool should support the ability to manage file permissions using the Windows Installer Lock Permission Lists in order to implement security permissions for directories, files, and Registry keys.
- The packaging tool should allow an administrator to digitally sign packages to ensure its fidelity.

8 REFERENCES

NMCI Release Development Deployment Guide (NRDDG)
http://www.nmci-sif.com/downloads/ReleaseDevelDeployGuide.zip
Novadigm Radia MSI Guidelines
http://techsupport.novadigm.com/kb/kb00609.asp
Microsoft Windows Installer: Benefits and Implementation for System Administrators
http://www.microsoft.com/windows2000/techinfo/administration/management/wininstaller.asp
Microsoft's Windows Installer Service Overview
http://www.microsoft.com/windows2000/techinfo/howitworks/management/installer.asp
Microsoft ORCA Utility
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msi/setup/orca_exe.asphttp://msdn.microsoft.com/library/default.asp?url=/library/en-us/msi/setup/windows_installer_development_tools.asp
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msi/setup/platform_sdk_components_for_windows_installer_developers.asp
Microsoft Platform SDK - Full Download with Local Install
http://www.microsoft.com/msdownload/platformsdk/sdkupdate/psdkredist.htm
http://www.microsoft.com/msdownload/platformsdk/sdkupdate/default.htm?p=/msdownload/platformsdk/sdkupdate/psdkredist.htm
http://www.microsoft.com/msdownload/platformsdk/sdkupdate/psdk-full.htm

Microsoft: Application Specification for Microsoft Windows 2000 for Desktop Applications

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnw2kcli/html/w2kcli.asp
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnw2kcli/html/w2kcli_appendixa.asp
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnw2kcli/html/w2kcli_chapter2.asp

Microsoft's "Certified for Windows" program

http://www.veritest.com/certification/ms/cfw/